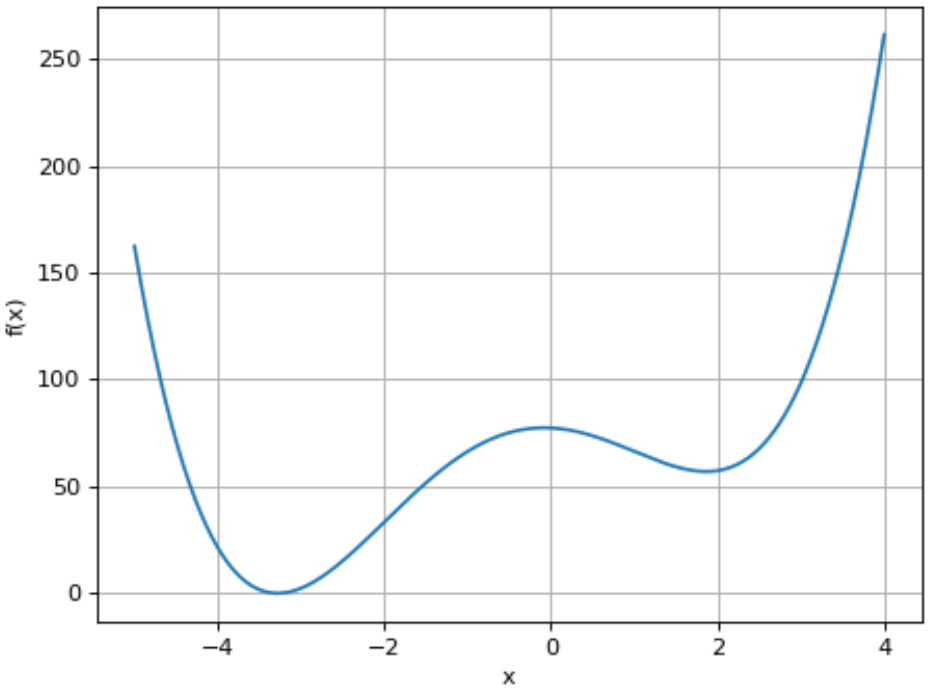
Recherche d'un minimum à une dimension

1. Introduction

Soit une fonction f d'une variable réelle et à valeurs réelles, admettant un minimum dans l'intervalle $]a, b[$, par exemple :

```
from matplotlib.pyplot import *
import numpy
def f(x):
    return x**4+2*x**3-12*x**2-2*x+77.37
a=-5
b=4
x=numpy.linspace(a,b,1000)
y=f(x)
figure()
plot(x,y)
xlabel("x")
ylabel("f(x)")
grid()
```



Il s'agit de déterminer le minimum de la fonction, appelé aussi minimum absolu, situé à $x \approx -3,3$. Cette fonction possède aussi un minimum relatif à $x \approx 2$.

La recherche d'un minimum consiste à partir d'un point quelconque et à suivre la pente descendante jusqu'à parvenir au minimum. En l'absence d'informations plus précises, le point initial de la recherche est choisi aléatoirement (avec une densité uniforme) sur l'intervalle $]a, b[$. Cependant, la recherche d'un minimum en suivant la pente descendante a autant de chances

de conduire au minimum relatif qu'au minimum absolu. Il faut donc un algorithme pour isoler un intervalle dans lequel se situe le minimum absolu, avant de faire une recherche plus précise de ce minimum. Nous utiliserons pour cela l'algorithme du *recuit simulé*.

2. Algorithme du recuit simulé

L'algorithme du recuit simulé est inspiré des méthodes de Monte-Carlo appliquées à la statistique de Boltzmann, et en particulier l'algorithme de Metropolis. Supposons que la fonction f définie ci-dessus représente l'énergie d'un système physique en fonction d'un paramètre x, définissant par exemple la structure cristalline d'un solide. D'après la loi de Boltzmann, la probabilité d'un état d'énergie E pour un système à la température T est :

$$p = p_0 \exp(-\frac{E}{k_b T}) \tag{1}$$

où p_0 est une constante et k_b la constante de Boltzmann.

Lorsque la température est suffisamment basse, le système se place au voisinage d'un état de minimum de l'énergie, soit le minimum absolu (état stable), plus rarement le minimum relatif (état métastable). Lorsque la température est très élevée, toutes les valeurs de x de l'intervalle sont équiprobables. Si le corps est refroidi lentement, il converge vers son état stable. Un refroidissement rapide peut le conduire dans un état métastable, c'est-à-dire un minimum relatif de l'énergie. Ce phénomène est mis à profit en métallurgie pour obtenir des structures cristallines métastables aux propriétés mécaniques intéressantes. En recuisant un cristal se trouvant dans un état métastable puis en le laissant refroidir lentement on peut l'amener dans son état stable.

L'algorithme du recuit simulé (*simulated annealing*) consiste en fait à simuler le processus de refroidissement très lent, en partant d'une température assez élevée. On devrait donc plutôt l'appeler algorithme du refroidissement simulé. L'algorithme permet de placer la variable x au voisinage du minimum absolu, mais pas de manière certaine. Avec un bon réglage de la vitesse de décroissance de la température, elle permet d'obtenir le minimum absolu avec une forte probabilité.

Pour notre problème, la loi de Boltzmann s'écrit :

$$p(x) = p_0 \exp(-\frac{f(x)}{T}) \tag{2}$$

Pour une température T donnée, l'algorithme de Metropolis consiste à générer une suite d'états (ici de valeurs de x) qui converge vers un ensemble d'états obéissant à la loi de probabilité de Boltzmann (ou une autre loi). On commence par une valeur x choisie aléatoirement (avec une densité de probabilité uniforme) dans l'intervalle $]a, b[$. La suite d'états est générée par l'algorithme suivant :

- ▷ Choix d'un nouvel état $x_1 = x + \Delta x$ ou $x_1 = x - \Delta x$. La direction est aléatoire mais le déplacement Δx est fixé au départ, assez petit par rapport à la largeur de l'intervalle.
- ▷ Si le nouvel état est plus probable que le précédent, c'est-à-dire si $f(x_1) < f(x)$, alors ce nouvel état est accepté.
- ▷ Si le nouvel état est moins probable que le précédent, il est accepté avec une probabilité égale au rapport des probabilités des deux états, c'est-à-dire $\exp(-(f(x_1) - f(x))/T)$. Pour ce faire, on tire un nombre réel à densité uniforme sur l'intervalle $[0, 1]$ et on le compare à cette probabilité.

Dans une simulation de Metropolis d'un système physique, on fait des calculs statistiques avec la suite des états ainsi générée. Dans l'algorithme du recuit simulé, seule la dernière valeur de x est utilisée. L'itération est répétée un grand nombre de fois avec la même température, puis on recommence avec une température un peu plus faible, en partant de la dernière valeur de x obtenue avec la température précédente. Avec une température finale assez faible, l'état final obtenu est très probablement très proche du minimum absolu.

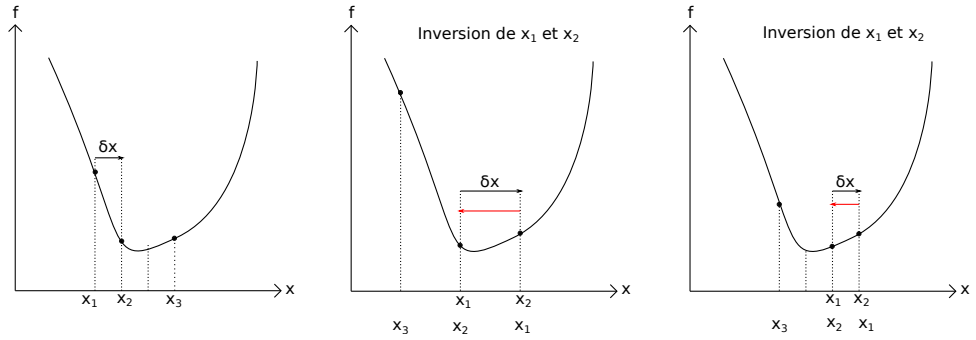
Cet algorithme nécessite des réglages qui dépendent du problème à résoudre : valeur de Δx, nombre d'itérations à chaque température, températures initiale et finale, loi de décroissance de la température.

3. Recherche du minimum par encadrement itératif

L'algorithme du recuit simulé nous donne une valeur x_1 proche du minimum absolu, du moins avec une forte probabilité. Pour rechercher avec précision le minimum, on peut à présent utiliser un algorithme déterministe.

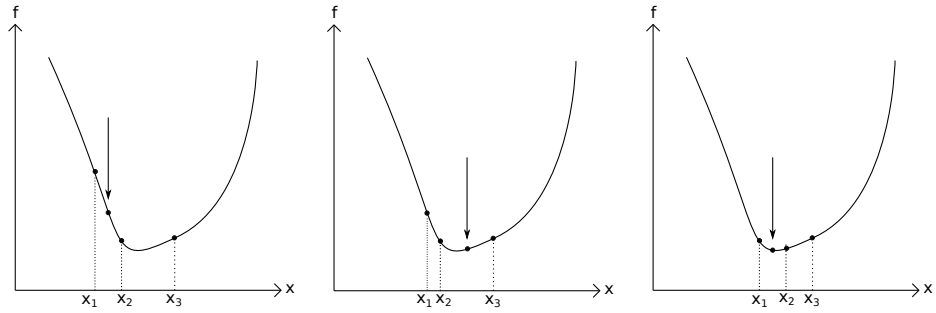
Il faut tout d'abord encadrer le minimum en cherchant x_1, x_2 et x_3 tels que $f(x_2) < f(x_1)$ et $f(x_2) < f(x_3)$, ce qui garantit la présence du minimum dans l'intervalle $[x_1, x_3]$.

Soit δx un déplacement assez petit (initialement positif). On considère tout d'abord $x_2 = x_1 + \delta x$ puis on échange éventuellement x_1 et x_2 pour que $f(x_2) < f(x_1)$ (auquel cas on change le signe de δx). En se déplaçant plusieurs fois de δx, on recherche ensuite une valeur x_3 telle que $f(x_3) > f(x_2)$. L'algorithme est illustré sur la figure suivante pour trois cas différents.



L'algorithme doit fonctionner quelle que soit la position de x_1 par rapport au minimum et quelle que soit la valeur de δx, à condition que celle-ci soit assez petite (en valeur absolue) pour qu'un déplacement de δx ne fasse pas sortir de la cuvette où se situe le minimum recherché.

On dispose à présent d'un encadrement du minimum par x_1 et x_3. On utilise une méthode itérative se répétant tant que $|x_3 - x_1| > \epsilon$, où ϵ est une tolérance égale au moins à la racine carrée de la précision des flottants. À chaque itération, une nouvelle valeur x est prise alternativement au milieu de l'intervalle $[x_1, x_2]$ ou bien au milieu de l'intervalle $[x_2, x_3]$. Cette valeur vient remplacer une des trois valeurs (x_1, x_2, x_3) de telle sorte que l'on ait toujours $f(x_2) < f(x_1)$ et $f(x_2) < f(x_3)$ afin de maintenir l'encadrement du minimum. Cet algorithme est illustré sur la figure suivante :



4. Implémentation

On utilise le module `random` et la fonction `random.random()`. Pour ne pas générer toujours la même suite de nombres aléatoires, on initialisera le générateur avec `random.seed()`.

[1] Écrire une fonction `recuit_simule(f,a,b)` qui effectue le recuit simulé pour la fonction `f` sur l'intervalle $[a, b]$. La rempérature pourra être variée de $T = 100$ à $T = 1$, avec une cinquantaine d'étapes comportant chacune un millier d'itérations. On pourra adopter le déplacement $\Delta x = 0.1$. La fonction renvoie x_1, la dernière valeur obtenue. Pour chaque température, faire un affichage de cette température et de la valeur de x finale.

[2] Tester plusieurs fois la fonction pour vérifier qu'elle donne le plus souvent une valeur proche du minimum absolu. Observer comment la valeur de x s'approche progressivement du minimu absolu.

[3] Écrire une fonction `encadrement(f,x1,delta_x)` qui effectue le premier encadrement du minimum et renvoie (x_1, x_2, x_3).

[4] Écrire une fonction `recherche_minimum(f,x1,x2,x3,tol)` qui effectue la recherche itérative par encadrement et renvoie le minimum. Tester avec une tolérance 10^{-4}.

[5] Mettre en place un test visant à déterminer la probabilité de trouver le minimum absolu et non pas le minimum relatif. Tester différentes vitesses de refroidissement.